We claim

1. A method for executing a sequential program in parallel on a system containing a plurality of processing nodes connected through an interconnection network, comprising the steps of:
   - segmenting the sequential program into multiple processes that are to be executed in parallel on different processing nodes,
   - establishing dynamic master slave relationships between processes to be executed in parallel based on the program flow requirements,
   - creating a job corresponding to each relation between a master and slave process,
   - creating and forwarding synchronous objects as arguments to each job, wherein said synchronous objects enable blocking of the master process only when the object is being accessed by the slave process and allow access by the master process at intermediate points of execution when the object is not being accessed by the slave process,
   - scheduling the job along with values of said arguments on a processing node and executing said job as a slave process on the processing node, and
   - receiving result of the job execution from the slave process on completion.

2. The method as claimed in claim 1 further comprising the step of providing automatic fault tolerance in the event of failure of a slave process, node or network involving:
   - storing object value during their transfer to the slave process in a buffer,
   - rescheduling the slave process on the same or on a different processing node, and
   - transferring the stored value of the objects from the buffer to the slave process.

3. The method as claimed in claim 2 wherein the object buffer is shared with buffer corresponding to the previous call to the parallel procedure, if object is added as non-returnable in the previous call and no modification to the object is made in between the calls.

4. The method as claimed in claim 1 wherein the step of creating and forwarding synchronous objects includes the use of references as arguments or return value to a parallel procedure.

5. The method as claimed in claim 1 wherein an executing slave process creates and forwards a new job for parallel execution on a different node together with synchronous objects as arguments, said new job executing as a slave process and the initiating process acting as the master process.

6. The method as claimed in claim 1 wherein an argument is of type 'returnable', when modifications made to the synchronous object in the parallel procedure are reflected back, and is of type 'non-returnable', when no modification is to be reflected back.

7. The method as claimed in claim 1 wherein said synchronous objects are derived from a special in-built wrapper class that is provided for encapsulating data synchronization features for each regular data type of the target language as well as custom data types defined by the programmer.

8. The method as claimed in claim 7 wherein the functionality of wrapper classes of any data type that is passed as an argument to a parallel procedure is added automatically at compile time.

9. The method as claimed in claim 1 wherein the step of scheduling the job at the selected processing node comprises the steps of:
   - providing the job information header,
   - providing the type information of each object added to said job
   - instantiating objects in the slave process from the type information provided, and
   - sending values of said arguments to the slave process as and when owned by the job.

10. The method as claimed in claim 1 wherein each synchronous object has an associated access control mechanism for ensuring that it is accessible only by the process to which it is currently available.

11. The method as claimed in claim 10 wherein the availability sequence of an object is

defined by an 'ownership queue' associated with said object, in which the 'current owner' of the object is the first element in the queue and subsequent elements are jobs awaiting ownership of the object, with ownership being transferred to each job in the sequence of the ownership queue and finally to the executing process, with said jobs being inserted in the queue if the object is added to them.

12. The method as claimed in claim 11 wherein on transfer of ownership of the object to a job, in which the object has been added as 'returnable', the object values are transferred to the corresponding slave process and the job remains the object's 'current owner' and after receiving the object value from the process corresponding to the 'current owner' the job releases ownership of the object.

13. The method as claimed in claim 11 wherein on transfer of ownership of the object to a job, in which the object is added as 'non-returnable', the object values are transferred to the corresponding slave process and ownership of the job is immediately released.

14. The method as claimed in claim 11 wherein the 'ownership queue' an object is linking of object's entry in the jobs to which object has been added in the order in which object ownership is to be transferred, with insertion of jobs in the queue involves only modifying the appropriate links.

15. The method as claimed in claim 11 wherein during said transfer of ownership, the jobs in which the object is added as 'non-returnable' are bypassed and the object value is transferred to the slave processes corresponding to the bypassed jobs.

16. The method as claimed in claim 1 wherein said object when passed as an argument to a parallel procedure can refer to other synchronous objects with the synchronization done over the value of referred objects also and the linkage structure maintained for each object in the slave process.

17. The method as claimed in claim 16 wherein an object reachable from an object added to the job as 'returnable' is also added to the job as 'returnable', while an object that is reachable from an object added as 'non-returnable' is added as 'non-returnable', with the value of the referred objects being synchronized by also adding them to the job.

18. The method as claimed in claim 16 wherein the linkage structure of objects is maintained by:

- making an entry of each referred object, which is added to the job, in the object information list in the job,
- storing the link information of objects within the object information list in the form of an index of objects' entry in the object information list,
- sending the object information list to the process corresponding to the job, and
- linking the newly created objects in the like manner.

19. The method as claimed in claim 16 wherein the linkage structure of the objects can be altered in the program with changes in linkage structure being reflected in a the processes where the object is transferred and synchronization being maintained in accordance with the new linkage structure.

20. The method as claimed in claim 19 wherein reflecting linkage structure changes and maintaining synchronization comprises the steps of:

- adding objects referred by a object only after ownership of the referring object is transferred to the job,
- maintaining the order of ownership consistent with the call order of parallel procedures in the master process,
- sending the type information of the referred objects and linkage information of the referring object to the process corresponding to the job as the referred object are added

to the job, and

- creating the objects from the received type information and linking them with the corresponding referring object there.

21. The method as claimed in claim 20 wherein maintaining the order of ownership transfer consistent with the call order of parallel procedure involves:

- holding the transfer of the ownership of a referred object till the ownership of the referring object gets transferred and its referred object added to the job thereafter, if the call order of the next owner of the referred object is greater than that of the referring object, and
- inserting the job, to which referred object is added, in between the ownership queue of the object such that job sequence in the ownership queue is consistent with the call order of the corresponding parallel procedures.

20. A system for executing a sequential program in parallel on a system containing a plurality of processing nodes connected through an interconnection network, comprising:

- a central processing unit,
- a system bus
- a communication unit connected to the system bus, and
- a memory connected to the system bus, containing:
  - means for segmenting the sequential program into multiple processes that are to be executed in parallel on different processing nodes,
  - means for establishing dynamic master slave relationships between processes to be executed in parallel based on the program flow requirements,
  - means for creating a job corresponding to each relation between a master and slave process,

- means for creating and forwarding synchronous objects as arguments to each job, wherein said synchronous objects enable blocking of the master process only when the object is being accessed by the slave process and allow access by the master process at intermediate points of execution when the object is not being accessed by the slave process,

- means for scheduling the job along with values of said arguments on a processing node and executing said job as a slave process on the processing node, and

- means for receiving result of the job execution from the slave process on completion.

23. The system as claimed in claim 22 further comprising means for providing automatic fault tolerance in the event of failure of a slave process, node or network by rescheduling the slave process on the same or different processing node.

24. A computer program product comprising computer readable program code stored on computer readable storage medium embodied therein for executing a sequential program in parallel on a system containing a plurality of processing nodes connected through an interconnection network, comprising:

- computer readable program code means configured for segmenting the sequential program into multiple processes that are to be executed in parallel on different processing nodes,

- computer readable program code means configured for establishing dynamic master slave relationships between processes to be executed in parallel based on the program flow requirements,

- computer readable program code means configured for creating a job corresponding to each relation between a master and slave process,

- computer readable program code means configured for creating and forwarding synchronous objects as arguments to each job, wherein said synchronous objects enable blocking of the master process only when the object is being accessed by the slave process and allow access by the master process at intermediate points of execution when the object is not being accessed by the slave process,

- computer readable program code means configured for scheduling the job along with values of said arguments on a processing node and executing said job as a slave process on the processing node, and

- computer readable program code means configured for receiving result of the job execution from the slave process on completion.

25. The computer program product as claimed in claim 24 further comprising computer readable program code means configured for providing automatic fault tolerance in the event of failure of a slave process, node or network by rescheduling the slave process on the same or different processing node.